# Investigations into partitioning of generalization processes in a distributed processing framework

**Frank Thiemann, Stefan Werder, Thomas Globig, Monika Sester**

Institute of Cartography and Geoinformatics, Leibniz Universität Hannover

**Abstract.** Processing of large spatial data sets often poses specific demands on computation time and allocation of resources. For an efficient processing, a partitioning of the whole task into subtasks is needed, which is mainly achieved by creating a spatial partitioning of the whole scene. When complex, context dependent operations have to be computed, such as generalization, a simple spatial partitioning is not possible. In the paper, different partitioning schemes will be presented for three different generalization operations. Furthermore, first results of using the parallelization framework Hadoop will be shown.

**Keywords:** Big data sets, Partitioning, generalization, parallelization

## 1 Introduction

Processing of large spatial data sets often poses specific demands on computation time and allocation of resources. For an efficient processing, a partitioning of the whole task into subtasks is needed, which is mainly achieved by creating a spatial partitioning of the whole scene. There are several partitioning schemes which can be applied (Werder and Krüger, 2009). A simple approach uses rectangular tiles, other approaches use the object structure in the data to separate different parts, e.g. the street and river network (Regnauld, 2001).

The general processing approach is composed of the following steps: cutting the whole data set into different adjacent parts, processing the parts separately and integrating the result. For simple applications such as raster operations in images, this method can be applied in a straightforward way. However, often operations have influence on neighboring tiles; then this has to be respected in the strategy. This is especially true for generalization operations which in general depend on the local spatial context.

The paper presents a framework which addresses two problems in generalizing large data sets: first of all, a strategy for tiling and combining the data in the whole process is presented. This strategy relies on the introduction of a buffer area around the tiles, which allows modeling the influence of the spatial context implicitly. The size of the buffer depends on the expected influence range of the operation. In the final step of combining the separate tiles, also the expected deviations during the processing have to be taken into account.

The second contribution of the paper is the processing of generalization operation in a parallel computing framework, in this case the Hadoop framework (White, 2012). The implications of applying the MapReduce algorithm are discussed concerning both distributed computation and storage. The possible options for the integration of legacy code into Hadoop are also discussed.

In the paper, three different generalization operations will be investigated: simplification, enlargement and aggregation of buildings (use case 3), simplification and aggregation of land-cover polygons (use case 2), and displacement (use case 1). The same underlying processing framework will be used; however, the approaches and the necessary parameters for the tiling and buffering will differ. Several experiments will be presented, together with an evaluation of the results both in terms of runtime and quality of the results.

## 2 Partitioning

As described above, spatial operations often rely on the inclusion of the spatial context, namely the influence of neighboring objects on the result. This is especially true for generalization operations.

It is not sufficient to only process the tiles separately and combine them afterwards, due to two reasons: firstly, objects can sit on the boundary of the tiles, thus they are split, or assigned to either neighboring tile. Secondly, most of the generalization operations do not operate on individual objects, but have a local influence or need local context. E.g. the generalization of buildings needs to analyze neighboring buildings to decide, if they should be aggregated; the aggregation of land-cover classes has to take the best neighbor into account; displacement does affect neighboring objects.

Thus, a partition scheme has to be devised, which includes the spatial context and generates partition results, which are identical at the boundaries of the partitions, in order to be seamlessly merged.

## 2.1 Spatial context

The spatial context can range from no influence to limited and infinite influence – depending on the objects and the spatial operations involved.

1) **No influence:** in the first simple case, object can be processed individually, thus an arbitrary partition can be used.
2) **Influence limited by natural breaks:** in this case, the knowledge about the objects involved can be used to identify limits of influence. A popular example is the road and river network, which e.g. limits the influence of aggregation operations: e.g. typically, buildings are not aggregated over roads or rivers. A second way to determine the fixed natural breaks is to use the given neighborhood range of an object or an operation. E.g. in the case of building aggregation, there is a maximum distance, beyond that buildings are no longer aggregated. Using this distance in a clustering process can determine a set of objects which can be processed independently.
3) **Influence limited by size of objects:** such a constraint can be applied in the case of aggregation and be defined by the maximum size of target objects.
4) **Unlimited influence:** one example for this case is the displacement operation, which as a potential unlimited influence, which, however, fades out rather quickly in larger distances from the displacing object. Another example is the influence of chaotic effects, which can, however, not be modeled and thus such processes are not suitable for partitioning.
5) **Random influence:** finally, the result of geospatial operations can depend on random influence, e.g. triggered by the underlying processes (e.g. Monte-Carlo-Methods, RANSAC). It has to be noted, that although all the random results are different, but still present correct solutions of the problem.

The following general sequence is applied:

1. Partitioning of the whole area into parts
2. Generalization of individual parts
3. Composition of parts to form whole area again

Depending on the type of partitioning scheme, the general sequence has to modified, which is described in the following.

## 2.2 Regular tiles

In order to take neighborhood and context into account, the tiles have to be enlarged by a buffer of a size, which depends on the type of generalization operation and the objects involved (see Figure 1, left).

Using this approach has the effect that the objects in the border regions also reside in other partitions, leading to the fact that they are processed more than once. The idea is, however, that for the area in the interior of the tile enough context is provided to take the correct generalization decisions. Areas outside the processed partition are considered too far away to influence the generalization result (Thiemann et al., 2011).

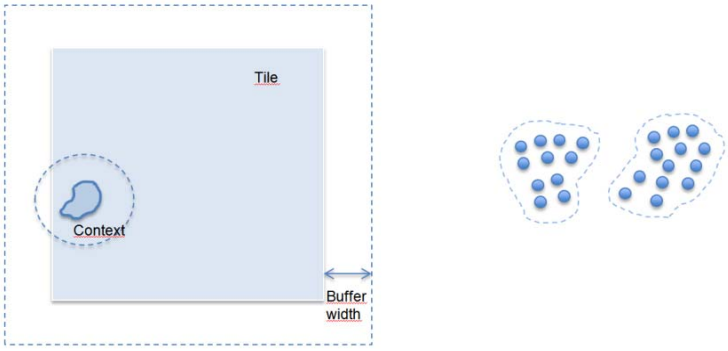


Figure 1: Left: regular tiling and buffering in order to take context of objects into account; Right: clustering of objects in given spatial neighborhood distance.

## 2.3 Clustering

In order to partition the data according to natural feature boundaries, a clustering has to be calculated, which is based on the maximum influence range (buffer) that objects can have on each other. In this way, a similar buffer size can be used as in the tile-case to determine appropriate clusters. Once the clusters are determined, they can be processed separately and then recombined just by appending the different data sets. Due to the choice of the buffer size, there should be no redundancy / overlap after the combination. Thus, the resulting partitions can be processed separately and recombined seamlessly (see Figure 1, right).

## 2.4 Composition

After the processing of the individual partitions the data have to be merged. When the tiles have been processed independently, the composition is straightforward, as there is no redundancy and no overlap. When a regular tiling has been used, objects on the boundary have been cut or they are have been processed in all neighboring tiles. It can be expected that areas in the border regions of the partition may be generalized incorrectly because of missing context. Therefore, the buffer area is clipped again to produce the original tile sizes, which then can be merged.

The composition of the results of the different parts can follow different strategies:

- **Selection:** as object in the border area have been processed twice, either object can be selected. A unique choice is to take the result which has its center in the tile. This approach is used for the composition of objects after building generalization with CHANGE, a program for building generalization developed at the of Cartography and Geoinformatics of the Leibniz Universität Hannover.
- **Cut and merge:** In case of discrete operations such as building generalization or land-cover aggregation, the effect of the generalization operation can be limited – in most cases the necessary buffer distance is determined by the size of the largest object. Then, objects on the boundary can be assumed to be the same and thus objects at the boundary are cut and merged with the adjacent partner object in the neighboring tile. This approach was used for the aggregation of land-cover polygons. It could also be used for the building generalization.
- **Match and adjust:** the assumption in case of displacement is that the effect is fading out, thus its influence will be continuously decreasing. In a certain buffer, the effect is expected to be small enough, not zero, but in the range of the mapping accuracy. Then corresponding objects are matched and possible geometric discrepancies are adjusted.

These methods can be used depending on the influence range of the operation and/or the objects.

In case the influence of the operation is limited, the border region has been selected appropriately, and the result is neither random nor chaotic, then neighboring tiles will produce the same result at their inner boundary. Then, either selection can be used, or cut and merge (use cases 2 and 3). In

case of infinite influence (use case 1), match and adjust is applied. In case of chaotic effects or only slowly fading influence, no general method can be proposed for the composition of the objects.

## 2.5 Quality measures

To study the influence of the buffer width on the quality and consistency of the result, measures to quantify the discrepancy of a solution with partitioning vs. a holistic solution have to be given. Alternative measures determine the degree of coincidence of corresponding features. With the help of these measures, appropriate buffer ranges can be determined.

In case of displacement, the displacement of the features at the tile boundaries is a direct measure (see Figure 2). To reconcile the small errors, the corresponding points on the offset are simply averaged, leading to a mean point on the boundary.

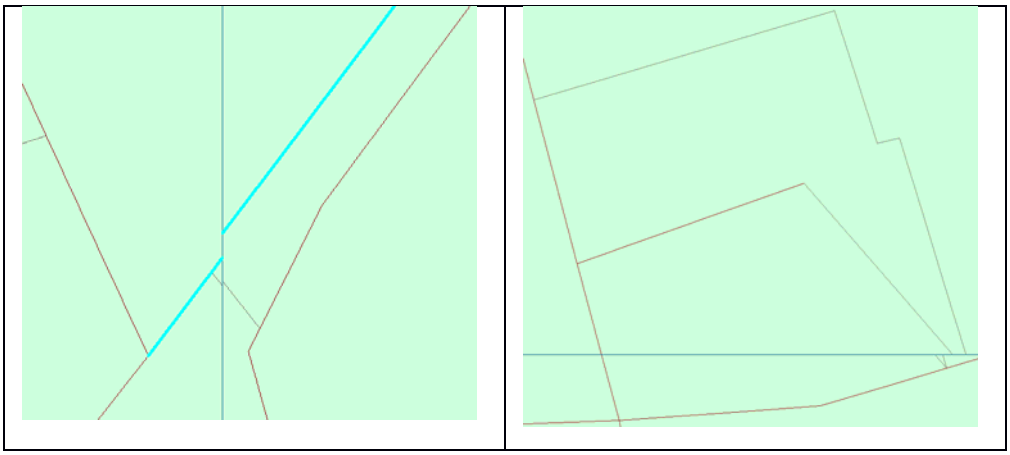

Figure 2: Displacement of lines (left) and polygons (right) after recombination at tile borders.

In case of land-cover aggregation, a consistent result can be expected, when the buffer is large enough. A measure of discrepancy resulting from too small buffer sizes is therefore the correct assignment of the land-cover class. Here, e.g. the size of the area with wrong land-cover class relative to the reference data set can be used as measure.

The generalization of buildings leads to correct results, when the buffer size is larger than the longest aggregated building block. The quality measure is the number of incorrect buildings. A problem is that aggregated buildings can get very large, e.g. industrial complexes of several kilometers. Therefore it is difficult to select an optimal buffer size without having a large processing overhead to determine the maximum aggregated area. Using the

cut and merge technique, however, the buffer size can be reduced to the local influence, corresponding to the gap tolerance for aggregating buildings plus the offset tolerance of the simplification of the building outlines. Then, as measure of the correctness the size of the area with missing or false building area can be used.

## 3 Parallelization with MapReduce

MapReduce is used to process the generalization in parallel on a cluster of commodity machines. MapReduce is a programming model and associated implementation invented at Google for simplifying the processing of large data sets (Dean and Ghemawat, 2004). Details of parallelization, fault-tolerance, data distribution and load balancing are hidden from the user, who is only responsible for the implementation of a sequential map and a sequential reduce function. MapReduce is based on the Google File System (GFS), a performant and highly scalable distributed file system, which is designed for data-intensive applications (Ghemawat et al., 2003).

Hadoop is an open source implementation of MapReduce and GFS. It is written in the Java programming language and derives in some aspects from the implementation at Google. White (2012) gives a detailed overview of Hadoop.

The map and reduce functions use key-value pairs as their input and output (see Table 1). The pairs are fed iteratively into the functions, i.e. in each call of the functions only a small partition of the input data is available. After all input pairs are processed, the values v2 with the same key k2 of all map function calls are grouped together and fed into the reduce function.

| Function | Input | Output |
|---|---|---|
| map | (k1,v1) | list(k2,v2) |
| reduce | (k2, list(v2)) | list(k3, v3) |

Table 1: Keys and values of the Hadoop map and reduce functions

The MapReduce workflow consists of several subsequent steps (see Figure 3). The input data is partitioned into so called splits. Each split is assigned to its individual map task. The tasks are automatically executed in parallel and run independently from each other. The output of the map tasks is then grouped into a user-defined number of partitions based on the keys k2. The

partitions are stored in temporary files and are moved to the reduce tasks. This step is called shuffle. The reduce tasks also run independently from each other and each produce a separate output file in the distributed file system.

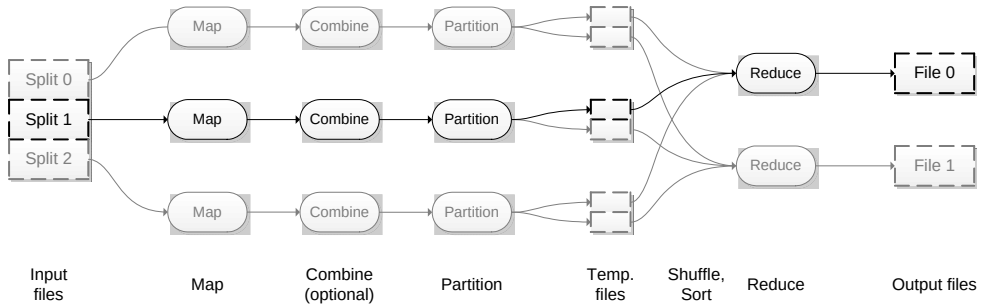


Figure 3: MapReduce workflow

At the ikg we installed a cluster of 5 work stations, each with a quad-core cpu and 16 GB RAM. In the MapReduce process 12 tasks are working in parallel.

# 4 Use Cases

## 4.1 Use Case 1: Displacement

PUSH is a program which performs a holistic displacement of all features of all geometric types in a scene. It is based on Least Squares Optimization (Sester, 2005). Although sparse matrix computation is used, there is a limit concerning the maximum number of points to process, which is mainly given by the available main memory. On a computer with 4 GByte main memory 360,000 unknowns, corresponding to 180,000 points can be processed in 3 min. This corresponds to a typical topographic map scene of approx. 15 x 15 km.

The following investigations have been conducted with a topographic data set of target scale 1 : 25,000. Different experiments were tested to determine an appropriate buffer radius, namely a radius, which is able to include all effects of a displacement operation. Displacement effects typically propagate into the neighborhood, with a decay effect. Thus, in the experiments the partitioning and generalization with different buffer sizes has been tested. As a measure to determine the buffer width at which the effect has faded away, the quality of fit of objects on the partition boundary after the recombination has been used. Displacements of lines or polygons are an indicator to what degree a separate displacement influences the shape and position of an object (see Figure 2).

The following results have been achieved for different buffer sizes. The total number of lines and polygons is 6981 and 5022, respectively. The maximum displacements of lines and polygons are listed. Also, the number of features which are displaced with a value larger than 1 cm is given.

| Buffer size [m] | Line displacement [m] | | Polygon displacement [m] | |
|---|---|---|---|---|
| | Max [m] | # >1 cm | Max [m] | # >1 cm |
| 1 m | 5,45 | 118 | 13,66 | 199 |
| 100 m | 1,61 | 109 | 3,55 | 167 |
| 500 m | 0,37 | 67 | 0,88 | 64 |
| 1000 m | 0,08 | 22 | 0,11 | 15 |
| 1500 m | 0,05 | 3 | 0,06 | 4 |

Table 2: Displacement at the border using different buffer sizes

In the target scale 1 : 25,000 0.1 mm corresponds to 2.5 m. Thus, all values below this threshold are not visible in the resulting map. A buffer size of only 1 m (i.e. using practically no buffer), leads to maximum displacements of 5.45 m for lines and 13.66 m for areas. These values decrease with increasing buffer size to 8 and 11 cm for 1000 m buffer and 5 and 6 cm for 1500 m buffer. It should be noted that among the 6981 lines only 22 have displacements larger than 1cm in case of a buffer of 1000 m, and only 3 when a buffer of 1500 m is applied. The remaining (small) discrepancies were removed by averaging the points on the offset on the tile boundary.

As a consequence, the buffer size of 1500 m was selected. With these parameters, an area of 38 km x 31 km has been processed. The tile size of 12 km with 1.5 km buffer leads to a processing tile of 15 x 15 km, of which 12 are needed to cover the whole data set. The (sequential) processing of the tiles took 29 min; the largest tile needed 3 min.

An extract of the solution of the displacement operation is given in Figure 4.

Figure 4: Situation before (left) and after displacement (right)

## 4.2 Use Case 2: Land-cover generalization

A generalization procedure for land-cover aggregation has been designed, which consists of aggregation, collapse and simplification (see Thiemann et al., 2011). To study the impact of the buffer width, experiments with a 45 x 45 km data set have been conducted. Results from a holistic generalization were taken as reference and compared to a series of results where the data set was split into four tiles and processed with different buffer widths, varying from 0 to 3.5 km. The comparison was based on the difference of wrongly assigned land-cover areas. Using no buffer leads to an error of approx. 2000 ha. At a buffer width of 2.5 km the error reduced to 0.16 ha (corresponding to 8.10 - 7 of the whole data set); at a buffer width of 3.5 km, no discrepancy was left. For the experiments a buffer width of 2.5 km was chosen.

In additional experiments the effect of the tile size on the computation time was also investigated. A data set of the whole state of Lower Saxony was processed. The run time of the three processes partitioning, generalization and composition for the different tile sizes is shown in Figure 5:. Reducing the tile sizes leads to a reduction of run time – however this stabilizes with a tile size of approx. 4000 km². The computations were made on an Intel Core 2 Duo (2.53 GHz) machine with 2 GB RAM with the approach, the whole land-cover data set of Germany could be processed in approx. 3 hours.

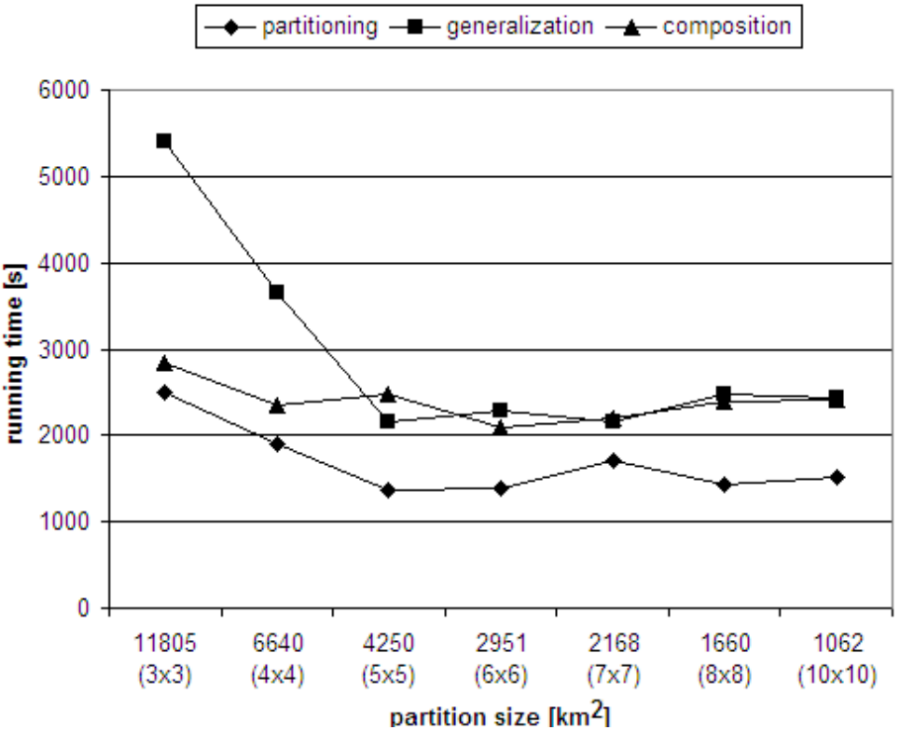


Figure 5: Running time for the land-cover aggregation of a 106,200 km² area using different partition sizes

## 4.3 Use Case 3: Building generalization with aggregation and simplification of building outline

### *4.3.1 Partitioning scheme*

For this generalization task the program CHANGE was used, which aggregates neighboring buildings (within a given distance) and simplifies the building outline. Due to the influence of the neighbors, the buildings cannot be generalized individually, but have to take the local context into account.

A tile based partitioning was applied; an example for the result at a partition border is given in Figure 6, right. It shows that in the immediate vicinity of the boundary, the results are identical – off the boundary they degrade depending on the degree of neighborhood. Thus, either a simple selection based on the centroid of the objects can be done; or objects on the

boundary line are cut and merged with the corresponding result on the other side of the boundary.

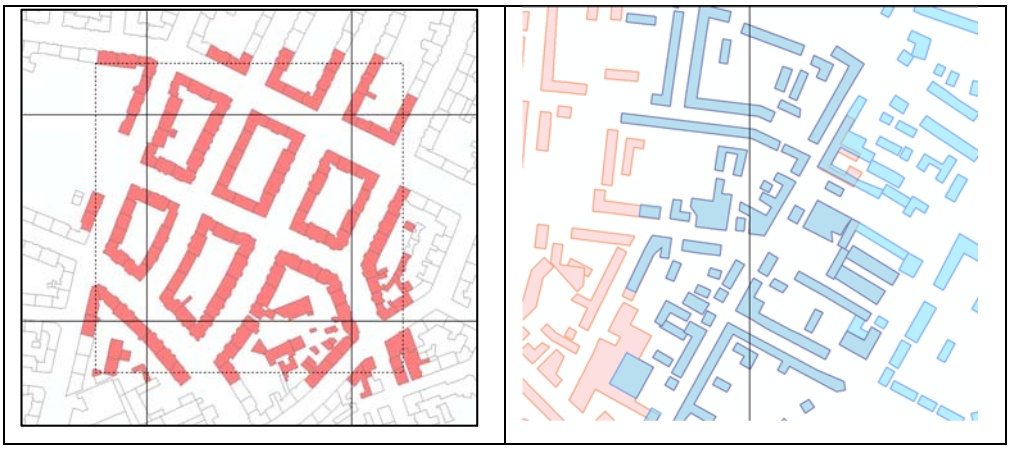

Figure 6: Left: Buffer area around central tile. Right: results in left and right partition.

As described above, the maximum influence for selecting complete buildings is the maximum size of a building in the result dataset. This size is limited but in some cases, there may be very large buildings. Choosing a too small buffer in this case leads to incomplete aggregations. Using the cut and merge method, however, the buffer size may be much smaller. Then, the buffer radius is only dependent on the generalization thresholds and not by the size of the resulting objects. This is due to the fact that the simplification operations (e.g. removal of small offsets) operate only very locally.

### *4.3.2 Parallel processing with Hadoop*

For the parallel processing of the data the Hadoop framework was used. The generalization software was installed on each cluster computer. The partitioning was created outside the Hadoop system using the tiling approach described above. The partitioned data was transferred in the ESRI shape format into the Hadoop file system. The size of each file must not exceed the 64 MByte of a Hadoop block, otherwise the binary data would be split by the file system without considering the data format, and thus destruct the geo-object structure.

Hadoop starts a mapper for each (shp) file. The init function of the mapper loads the other necessary files (shx, dbf, config files) in the local file system of the cluster computer. Then, the generalization software is executed. After

the generalization software has finished the resulting files are written back into the Hadoop file system. The reduce step is not used. The combination of the partition is also done outside of Hadoop framework.

Very large data sets could be processed: The first is a free data set of Boston containing 230,000 buildings, which was divided into 53 partitions of 4000 m x 4000 m plus a buffer of 200 m. This data could be processed in 8 minutes. In comparison to a non-parallel processing using 31 minutes, this is a speed up of approx. 4 times. The second data set of Chicago, consisting of 800,000 buildings, could not be processed in a sequential way due to memory demands required by the huge data set. Using the parallel processing in Hadoop, 124 partitions were created (in sum 250 MB), which could be processed in 18 minutes.

It should be noted, that there is a linear and a constant offset for the job management. In the above examples, this was a factor of 2.9 and a constant offset of 6 seconds for each partition.

In the implemented processing scheme, the MapReduce was not used. The partitioning and composition of the results was done outside the framework. Thus, Hadoop was used to split the partition list, distribute the tasks to the different cluster computers, and gather the results. The benefit of this approach is that the processes and the data are automatically distributed and controlled, processes failing to finish are restarted, and the sequence is prioritized according to the job size.

## 5 Discussion and Outlook

In the paper, different possibilities for the processing of very large spatial data sets have been proposed. In order to take the typical spatial context of objects and operations into account, an adequate partitioning scheme has to be devised. In three different use cases with different characteristics it could be shown that a partitioning is indeed possible and leads to the possibility of processing large data sets, and a speed up of the generalization operations. First experiments with the Hadoop framework proved its usefulness. In the future, also the other generalization operations are planned to be applied in this framework.

Generalizing large data sets in a parallel way can not only be used for the processing of large data sets, but also for update. Also there, the spatial context of an update situation has to be taken into account when including it in the data set. One option to determine the maximum range of an update operation is the (incremental) use of neighboring information in an MRDB context (Haunert & Sester, 2005). Another option is to determine the

influence range using the buffer approach as presented in this paper: instead of a regular tile, the update object is buffered, generalized separately, and combined with the data set again.

## 6 References


Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, Proceedings of the 6th Symposium on Operating System Design and Implementation, San Francisco, CA, USA, 2004.

Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google File System, Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 2003.

Haunert, J.-H. and M. Sester: Propagating updates between linked data sets of different scale, Proceedings of the 22nd International Cartographic Conference (ICC'05), La Coruna, Spain, 2005

Regnauld, N.: Contextual Building Typification in Automated Map Generalization, Algorithmica 30: 312–333, Springer-Verlag New York Inc, 2001.

Sester, M.: Optimizing Approaches for Generalization and Data Abstraction, International Journal of Geographic Information Science, vol. 19 Nr. 8-9, p. 871-897, 2005.

Thiemann, F., H. Warneke, M. Sester and U. Lipeck: A Scalable Approach for Generalization of Land Cover Data, Advancing Geoinformation Science for a Changing World, p. 399 - 420, Heidelberg, 2011.

Werder, S., Krüger, A.: Parallelizing Geospatial Tasks in Grid Computing, GIS.Science 3, p. 71-76, 2009.

White, T.: Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media, Sebastopol, CA, USA, 2012.